DPDK Community Survey

Signed-off-by: Honnappa Nagarahalli <honnappa.nagarahalli@arm.com>

Reviewed-by: Bruce Richardson <bruce.richardson@intel.com>

Acked-by: Maxime Coquelin <maxime.coquelin@redhat.com>

May 7, 2019

Introduction

There have been discussions in the past to change the DPDK development process to make use of tools such as Gerrit, GitHub pull requests etc. The discussion was brought up again during the 2018 North America DPDK summit. Subsequently the DPDK technical board decided on 30th Jan 2019 to conduct a survey to understand the needs of the community. It was decided to conduct a 2 part survey. The first part was for understanding the actual pain points of the development process the community faces. The second part, which was contingent on the outcome of the first survey, was intended to receive feedback on the new set of tools and methodologies.

Survey

The first survey was sent out on 27th Feb 2019 after the questions were deliberated by the tech board and other volunteers. 27 responses were received in the first week. After a reminder was sent out, a total of 41 responses were received by the first deadline of 13th March 2019.

After seeing the poor response, the technical board took a decision to extend the deadline till 23rd March 2019. It was also decided to send the survey to VPP and SPDK communities as well. Linux Foundation also announced the survey on Twitter.

Finally, 59 responses were received indicating not a huge interest in changing the current process.

There are 119 authors with at least 10 patches since 18.02 release. Out of them, 92 (77%) did not take the survey.

Results

This section covers the questions asked and the assessment of the responses.

1. Ease of accessing the code

   93% found it easy to access the code. The rest suggested following changes.

   a. The subtrees are not advertised very well. It is difficult for a newbie to imagine that such trees exist. Add the subtree list to dpdk.org.
   b. Having a website similar to the kernel elixir server, https://elixir.bootlin.com/linux/latest/source would be helpful.

2. Do you find the coding guidelines easy to follow?

   90% found it easy to follow the coding guidelines. The rest suggested the following changes.

   a. The guidelines are split across multiple links, making it difficult to follow.
   b. A tool to enforce coding style and tool to automatically format the code will be helpful.
   c. There is no way to verify if the 'Fixes' footer is correct.
   d. Process for integrating changes when the reviewers/gatekeepers don't provide timely feedback. This is resulting in keeping local forks as their improvements could not be upstreamed.
   e. The practice of breaking a new library/driver into many, separate patches has the opposite effect of the intention. It also requires extra work from the author.

3. Ease of using the scripts in 'devtools'

   78% found it easy. 22% responded with issues.

   a. 41% (of 22%) pointed to difficulty in setting up checkpatches.sh. It requires kernel tree and setting up environment variable pointing to checkpatch.pl. The steps mentioned in the guidelines are missing some information. Moving to different systems result in significant effort to setup the environment again.
   b. The tools running upstream report different problems due to using different kernel version locally.
   c. The scripts should solve the problems, at least for common problems like white spaces.
   d. There is no consistency between the parameters across scripts.
   e. test-build.sh/test-meson-builds.sh are not obvious to setup. Trying to ensure that a patch compiles on different arches, different compilers, shared lib, debug, and with all the various dependencies enabled is difficult.

    f. A proper CI integrated with github that automatically shows the results in each github PR would be better.

4. Ease of sending patches.

   80% found it easy. 20% responded with issues.

    a. 58% (of 20%) indicated the process is too cumbersome, especially for infrequent contributors. One of the respondents says sending patches is a nerve-wrecking process to make sure nothing is missed. Suggested providing a cheat-sheet.
    b. Dealing with mboxes via patchwork is much more laborious than just pulling a branch from a GitHub/GitLab fork when wanting to try a series that is not merged yet.
    c. Sending multiple revisions is a hassle when compared to fork + pull request of GitHub/GitLab
    d. Documentation is not clear on how to properly thread subsequent versions of patch sets. Some examples in the documentation would be helpful. One respondent, who mentioned the process is easy, talked about effort required in ensuring threading.
    e. Couple of respondents mentioned that they have moved from email to Gerrit and they are liking it (no need to add the reviewers, message id etc each time).

5. Is the current level of automation helpful?

   73% say yes. 27% responded with suggestions.

    a. A single script, with no options or setup, that runs all that is required and indicates if the patch is ready to send. Alternatively, a sand box system like Travis CI which indicates if all the checks/validations have passed. This will also make sure that patches have adequate essential checks already done.
    b. More of - tests on Arm platforms, performance tests, better coverage on basic checks, simple testpmd runs, applications with real traffic.

6. Do you get enough reviews on the patches?

   61% said yes, 39% said no.

7. Is the rate of response enough?

49% said yes, 51% said no.

8. Does patchwork help you in managing the patches?

   80% say yes. 20% responded with issues.

   a. Does not integrate with the process well.
   b. No way to link revisions of the patch, no way to get notified about patches under one's maintainership, relies a lot on manual process (such as acks etc), difficult to find one's own patch (no direct link, need to search), hard to manage patch sets that have dependency.
   c. It will be better if patchwork can automatically mark obsolete patch as superseded when new patch version is posted. Also, prefer to be able to leave and reply to comments online.

9. Do you find email based review helpful?

   73% say yes. 27% responded no.

   Out of the 27% 'no responses'

   a. Slow responses - 22%
   b. Comments lost or not addressed - 34%
   c. Managing the large list of emails is difficult - 44%

10. 26 responses were received for unstructured comments/feedback. Following are some of the prominent themes.
    a. Comments for Gerrit/GitHub/GitLab - 12 (Advantages over email, ability to see more lines around the changed lines, sending multiple revisions is easy)
    b. Comments for continuing with email - 4 (Provides ability to work offline, mail-list archive is helpful to track, appear in google search)
    c. Core code gets enough reviews, libraries do not get enough reviews.
    d. Although we see features getting added and announced, when we actually review the implementation of the features, we typically find missing core components, or implements which aren't suitable for our use. We stay away from the more "interesting" features since we don't have the time to evaluate whether the feature will meet our requirements.

e. Need to know earlier if a feature will/will not be merged, rather than finding out when it doesn't make an RC. A public list of patches that are being considered with a status of OnTrack, AtRisk, NotThisTime would be helpful.
f. Need for a ‘Linus Torvalds’ for DPDK who keeps going over pending patches and discussions and saying a final word or providing feedback on what is missing for acceptance. That person could assign people to lost questions from users at the same time.
g. Subject line of the commit messages requires more standardization. For ex: if the patch is about enabling a feature in hash library, fixed format would look something like: 'lib/hash: feature, (feature name)'.
h. DTS requires significant improvements.

Conclusion

Looking at the responses for questions 8, 14 and 16 (4, 8 and 9 respectively in this summary report), it is apparent that the community is happy with the current process. Hence the technical board took a decision to not send out the second survey.

Suggestions for consideration

Even though the outcome of the survey is to stay with the current process, following are some concerns/suggestions that the DPDK Tech Board should review. Note that these are the concerns expressed by the minority of the respondents.

1) The subtrees are not advertised very well. It is difficult for a newbie to imagine that such trees exist. Add the subtree list to dpdk.org (or any prominent place).
2) Having a website similar to the kernel elixir server, https://elixir.bootlin.com/linux/latest/source would be helpful.
3) A clear process should be defined for integrating changes when the reviewers/gatekeepers don't provide timely feedback. This is resulting in keeping local forks as their improvements could not be upstreamed.
4) It is difficult to setup checkpatches.sh. It requires kernel tree and setting up environment variable pointing to checkpatch.pl. The steps mentioned in the guidelines are missing some information. Moving to different systems result in significant effort to setup the environment again.

   There is no consistency between the parameters across scripts (some use -1 and some use -n 1). test-build.sh/test-meson-builds.sh are not obvious to setup. Trying to ensure that a patch compiles on different arches, different compilers, shared lib, debug, and with all the various dependencies enabled is difficult. A proper CI integrated with github that automatically shows the results in each github PR would be better.

5) Dealing with mboxes via patchwork is much more laborious. It is not clear which tree the patch should be applied.
6) Documentation is not clear on how to properly thread subsequent versions of patch sets. Some examples in the documentation would be helpful.
7) 39% of the respondents said they do not get enough reviews. May be the maintainers should delegate some reviews to other regular contributors?
8) Issues faced with patchwork - no way to link revisions of the patch, no way to get notified about patches under one's maintainership, relies a lot on manual process (such as acks etc), difficult to find one's own patch (no direct link, need to search), hard to manage patch sets that have dependency.

   It will be better if patchwork can automatically mark obsolete patch as superseded when new patch version is posted. Also, prefer to be able to reply to comments online.